Recurrent Reinforcement Learning for Lander Control during Powered Descent and Landing

Marco Anselmi
Student Number: 160623970
Project Supervisor: Dr. Angadh Nanjangud
School of Electronic Engineering and Computer Science
Queen Mary University of London
London, UK

***Abstract*—Future missions into space to planetary bodies will need to have accurate guidance, navigation and control during the powered descent phase, such that it will be possible to target a specific location on the surface of the planetary body, and achieve pinpoint landing accuracy. To achieve this, it requires a navigation system capable of identifying the lander's state in real time, and a guidance and control system capable of mapping the lander's state to commands for each of the lander's engines. In this paper, an integrated guidance and control technique, based on a Recurrent Reinforcement Learning algorithm, is presented. This technique uses both Reinforcement Learning and Recurrent Neural Networks to learn a policy that maps the lander's state to a thrust command for each lander engine. Specifically, the Reinforcement Learning technique used to learn the policy is Proximal Policy Optimization. The presented results analyses the performance of the propoosed method when trained for 250,000 timesteps and 500,000 timesteps, in the continuous Lunar-Lander environment provided by OpenAI Gym.**

***Index Terms*—Reinforcement Learning, Recurrent Neural Networks, Recurrent Reinforcement Learning, Guidance and Control System**

I. Introduction

Space missions to planetary bodies have always needed some form of landing system. From the Luna 9 landing, which slowed the spacecraft with retrorockets and then ejected the landing capsule at 5 meters from the surface; to the most recent landing on Mars of the Perseverance Rover, which uses a camera system to select a landing spot before using rockets to slowdown to a speed of 1.7 miles per hour, once it reaches a height of about 66 feet it then lowers the rover with cables, such that when the rover has landed the rest of the spacecraft can detach and go perform its uncontrolled landing a safe distance away from the rover. As we continue the exploration of planetary bodies, highly accurate landing systems will be required to ensure the safety of both the contents of the spacecraft and the spacecraft itself.

Currently used techniques and most proposed guidance and control methods use two separate and independently optimized systems for guidance and control. However, having two separately optimized systems can be limiting, as many complications can arise in the integration of the two systems. First, the reference trajectory of the system must ensure that the acceleration commands are executable given the thrusters configuration on the lander. Having separate guidance and control can also make the design and development processes much more difficult, as many issues might only arise when the systems are combined in simulations, therefore, requiring the redesign and modification of one or more of the systems.

An alternative to the split systems is the Integrated Guidance and Control (IGC). Recently, IGCs for lander control during missions to planetary bodies have been proposed by Zhang and Yang in [35], in which they proposed a system that uses a disturbance observer and multiple sliding surface techniques for a Mars landing; by Wibben and Furfaro [34], who for a lunar landing developed a Multiple Sliding Surface Guidance and Control (MSSGC) algorithm; and by Gaudet, Linares and Furfaro [10], who used a Reinforcement Learning (RL) Technique called Proximal Policy Optimization (PPO) to learn a policy that maps the lander's estimated state directly to a commanded thrust for each engine, with the policy resulting in accurate and fuel-efficient trajectories.

RL techniques, pose an interesting solution to the guidance and control problem, as they can simplify the inclusion of constraints on the terminal condition that the spacecraft should be in. RL techniques include constraints by means of the reward function, by shaping the rewards that the agent receives, through its interactions with the environment, it is possible to shape the behaviour of the agent. However, they also have many defects that discourage their use in real life settings. Firstly, there is no guarantee that the trajectories obtained by the policy are optimal, although it is possible to get close to optimal performance by defining a good reward function. Secondly, classic RL techniques rely on the assumption that the information they receive from the environment is markovian, while in real life most of the times it is not, meaning that the future states and rewards realy on more than just the system's current input; therefore, the environments become Partially-Observable Markov Decision Processes (POMDPs) from Markov Decision Processes (MDPs).

In this project the aim is to develop an integrated guidance and control system by devising a technique that combines Reinforcement Learning and Recurrent Neural Networks. The recurrent part of the system will be used to improve the performance of the RL technique and as a solution to the POMDPs problem.

II. Related Work

A. Recurrent Reinforcement Learning

Recurrent Reinforcement Learning (RRL) techniques, are a set of techniques that aim to integrate Recurrent Neural Networks (RNNs) together with Reinforcement Learning (RL). The aim of RRL techniques, in control theory, is to be a solution to the problem of POMDPs. This is because Recurrent Neural Networks are capable of storing information regarding a previously encountered state, thus, by utilizing that information it is possible to estimate the systems' dynamics and internal states, thus reconstructing the Markovian state space.

One of the earliest, if not the first, Recurrent Reinforcement Learning technique, came out in 1996, when Moody and Wu [27] proposed a Recurrent Reinforcement Learning algorithm for the optimization of Trading Systems and Portfolio. Ever since then, it has mainly been used in the field of trading [7], [3], [12], [11]. Very few applications of RRL have been done in other fields.

One of the first integrations of Reinforcement Learning and Recurrent Neural Networks, outside the field of trading, was done by Bakker in [5], and in [4]. In these two papers, Bakker combines a Long-Short Term Memory (LSTM) network, which is a kind of Recurrent Neural Network, together with an actor/critic RL model. Similar to Bakker, in [15] Hausknecht and Stone have developed a RRL technique that combines Deep Q-Learning with LSTM. In their work, they developed a Convolutional Neural Network to be used as the Deep Q-Learning network, and substituted the last fully-connected layer with a LSTM layer.

While the main aspect on which researchers have focused is on the RL aspect of the technique and the usage of RL learning methods to teach neural networks, there has also been some research on the role of the RNN. In [29], Schafer analysed the use of RNNs as State Space Models, in which, the hidden state of the network could be used to represent the current state of the system and as the input to the Reinforcement Learning part. Schafer explored the use of such RNNs in two different cases. In the first case the RNN and RL parts are separate from each other and are also trained separately. In the second scenario, the RNN and RL parts are unified under a single network, with each of the two parts of the network trained separately.

Similar to the work presented in [29], Li et al. [17] proposed a technique that incorporated both the RL model and the RNN into one network. In Li et al.'s method, a RNN is used as a state space model that given the current agent's observations of the state, it can predict the next set of observations and the reward. The RL algorithm Q-Learning is then attached to the hidden layer of the RNN, and uses the hidden layer as its input. The main difference between [17] and [29] is that [17] has developed a method to train both parts of the network at the same time, each with its own strategy. The RNN part is trained using a supervised learning approach, in which the error is given by the mean squared error with the true observations and rewards. The RL part is trained using normal RL approach, in which its aim is to maximise the reward received.

B. Autonomous Guidance Methods

Autonomous Guidance Methods (AGMs) are used for the planning of flight trajectories and/or providing in real-time commands to guide a vehicle to its desired destination, while adhering to constraints and terminal conditions. AGMs can be divided into three categories: Analytical Guidance Methods, Numerical Optimization Methods and Learning Methods.

1) Analytical Guidance Methods

Analytical Guidance Methods were the earliest methods to be implemented. These methods aim to derive the optimal control law by simplifying the general powered landing problem such it doesn't consider factors such as aerodynamic drag, variable mass of the rocket and changing gravitational field.

Polynomial guidance:
The Polynomial guidance method, implemented as the Apollo guidance law, and described by Klumpp in [16], it considers the acceleration of the vehicle as a quadratic polynomial with respect to time, from which, the velocity and position can be obtained by integration. Eq. 1 shows the sets of equations that are used as the basis for the formulation of the Polynomial guidance method.

$$\begin{cases} a(t) = C_0 + C_0 t + C_2 t^2 \\ v(t) = v_0 + C_0 t + \frac{1}{2} C_1 t^2 + \frac{1}{3} C_2 t^3 \\ r(t) = r_0 + v_0 t + \frac{1}{2} C_0 t^2 + \frac{1}{6} C_1 t^3 + \frac{1}{12} C_2 t^4 \\ a(t_f) = a_f, v(t_f) = v_f, r(t_f) = r_f \end{cases} \tag{1}$$

From this set of equations, we can derive the following equations.

$$\begin{cases} C_0 = a_f - \frac{6}{t_f}(v_f - v_0) + \frac{12}{t_f^2}(r_f - r_0 - \dot{r}_0 t_f) \\ C_1 = -\frac{6}{t_f} a_f - \frac{30}{t_f^2}(v_f - v_0) + \frac{48}{t_f^3}(r_f - r_0 - \dot{r}_0 t_f) \\ C_2 = \frac{6}{t_f^2} a_f - \frac{24}{t_f^3}(v_f - v_0) + \frac{36}{t_f^4}(r_f - r_0 - \dot{r}_0 t_f) \end{cases} \tag{2}$$

The vector of the coefficients $[C_0, C_1, C_2]$ is the variable that needs to be optimized.

These sets of equations, only have the terminal position, velocity and acceleration as constraints. By developing higher order polynomial equations, it is possible to introduce more constraints as well.

Guidance methods based on the maximum principle:
In 1964 Meditch in [26] introduced a guidance method based on the Pontryagin maximum principle, in which an optimal thrust program is found. The optimal thrust program is defined

as the thrust program that can bring the vehicle from an initial state to the terminal state and can minimize equation 3.

$$S = -\int_0^\tau \dot{m}(t)dt = m(0) - m(t) \tag{3}$$

Where $\dot{m}(t)$ is the mass flow rate and $m(t)$ is the total mass of the vehicle at time t.

The optimal thrust program can be synthesized by determining an appropriate switching function, which, in turn, consists in determining the relation shown in equation 4, such that if the maximum thrust is applied from the moment the relation is met to the moment the vehicle lands, a soft landing is achieved.

$$f(x_1, x_2) = \frac{b}{a}r_1 + 2a\sqrt{\frac{r_1}{a}} + r_2 = 0 \tag{4}$$

Where $a = \frac{1}{2}(\frac{k\alpha - gM_0}{M_0})$, $b = \frac{k\alpha^2}{2M_0^2}$, k is the velocity of the exhaust gases with respect to the vehicle, g is the acceleration of gravity and x_1, x_2 and M_0 are the altitude, altitude rate and mass at the initiation of thrusting.

ZEM/ZEV feedback guidance law:
ZEM and ZEV represent the position and velocity errors between the lander and the target at time t_{go}, if no control force is applied, where t_{go} is the time-to-go until intercept with target (i.e $t_{go} = t_f - t$). Respectively:

$$\begin{cases} \mathbf{zem}(t) = r_f - r(t) + t_{go}v(t) \\ \qquad\qquad + \int_t^{t_f}(t_f - \tau)g(\tau)d\tau, \\ \mathbf{zev}(t) = v_f - v(t) + \int_t^{t_f} g(\tau)d\tau \end{cases} \tag{5}$$

The ZEM/ZEV guidance law ([8], [13], [36], [37]) can be obtained by minimizing a performance index Eq. 6.

$$J = \frac{1}{2}\int_t^{t_f} u^T(\tau)u(\tau)d\tau \tag{6}$$

By using optimal control theory and the ZEM/ZEV functions, we can then obtain the optimal acceleration command as:

$$u = \frac{6}{t_{go}^2}\mathbf{zem} - \frac{2}{t_{go}}\mathbf{zev} \tag{7}$$

2) Numerical Optimization Methods
Modern embedded computers are capable of solving nonlinear programming problems using a Numerical Optimization Algorithm (NOA). The idea behind NOAs is to find the optimal solution to a problem, by iteratively calculating the direction of the solution and the step size to take in the calculated direction. The main factor that affects the convergence of the algorithm, is the convexity of the problem. If the problem is convex, convergence is theoretically guranteed. Because of this the most popular NOAs are the Convex Optimization Methods (COM). Convex optimization, however, is not applicable to all problems, since, the convex sets, constraints and means of lossless convexification are limited. Altough, non-convex optimization algorithms exist, in order for them to be used for online programming, suitable initial values need to be selected. Other NOAs include the Homotopy Methods, the Sensitivity Methods

Convex Optimization Methods:
Convex Optimization Methods are a set of optimization algorithms that require a convex objective function, linear equality constraints and convex inequality constraints. COMs can be divided mainly into three categories of algorithms, Linear Programming (LP), Second-Order Cone Programming (SOCP), Semi-Definite Programming (SDP), all of which require fixed endpoints to the independent variables.

COMs are well suited to solving control problems with linear dynamics and convex constraints for online programming, as they can solve convex problems in polynomial time. However, most real-life problems are not inherently convex and thus, not readily solvable. Research has thus been focused on convexifying non-convex problem, such that they can be solved by COMs. For the powered descent and landing problem, the linearity of the dynamics can be obtained by assuming constant gravitational acceleration and ignoring aerodynamic forces [2]. Furthermore, for certain non-convex control constraints, they can be replaced by convex ones without loss of generalization via the use of a technique called ”Lossless Convexification” [1], [6], [14], [24].

However, if the change in gravitational acceleration and the aerodynamic forces need to be considered, the convexification of the problem becomes much harder. To solve for the non-linearity of the dynamics a successive convex programming approach was introduced by Lu and Liu in [19], and was then applied to the landing problem [32], [18]. Mao et al. [25] then proposed another successive convexification algorithm that made use of virtual control and trust regions to aid the convergence. The successive convex programming approach has also been successfully applied to the 6 Degrees-of-Freedom problem [33], [31].

Homotopy Method:
Traditional Newton-based methods used for optimization often have difficulties in converging to the solution, because their convergence relies on an good initial guess. The purpose of the homotopy method, is to overcome the reliance on a good initial guess. The basic concept of homotopy methods is to transform the problem of solving algebraic equations $f(x) = 0$, into solving a new set of equations, $h(x, \rho) = 0$, where the parameter ρ is the homotopy parameter, and h is defined as $h(x, \rho) = \rho f(x) + (1-\rho)g(x)$. As ρ changes from 0 to 1, a solution path $x^*(\rho)$ is created. The goal of the homotopy method is to follow the solution path from $x^*(0)$ and reach $x^*(1)$, which is the solution to the original problem. [22] and [20] both utilize a homotopy based backtracking strategy to select good initial values and to help solve the trajectory optimization problem. By setting the trajectory optimization problem to:

$$
\begin{aligned}
A(x,u,t) : min\ \Phi(z(t_f)),& \\
s.t.\ \dot{z} = f(z(t), y(t), u(t)), z(t_0) = z_0,& \\
g(z(t), y(t), u(t)) = 0,& \\
u_L \leq u(t) \leq u_U,& \\
\psi(z(t_f)) \leq 0&
\end{aligned} \tag{8}
$$

We can turn the problem into an homotopy function as follows:

$$
\begin{aligned}
L(x,u,\rho) = A(x,u,t,\delta(\rho)) \\
\delta(\rho) = \rho\delta(1) + (1-\rho)\delta
\end{aligned} \tag{9}
$$

Then we can solve the problem by following the homotopy backtracking strategy, shown in Algorithm 1, until the optimization converges.

Algorithm 1 Homotopy Backtracking ALgorithm

```
1: Step 1: Set the current homotopy parameter $\rho_c = 0$, target homotopy parameter $\rho_t = 1$, $(x_0, u_0) = (x_{set}, u_{set})$ (given initial values).
2: Step 2: Solve $L(x, u, \rho_t)$ with the initial values $(x_0, u_0)$. If successful, go to Step 4; otherwise, proceed to the next step.
3: Step 3: If $\rho_t - \rho_c \geq \epsilon$ (a predefined number), calculate $\rho_t = \rho_c + (\rho_t - \rho_c)/2$ and go back to Step 2. Otherwise, the backtracking procedure fails to stop.
4: Step 4: If $\rho_t \neq 1$, then $(x_0, u_0) = (x(t), u(t), t, \rho_c), \rho_c = \rho_t, \rho_t = 1$, and go back to Step 2. Otherwise, the procedure is successfully terminated.
```

Sensitivity Method:
Sensitivity methods are based on the use of sensitivity analysis for NLP. This analysis provides information on regularity and curvature conditions at Karush-Kuhn-Tucker (KKT) points, assesses which variables have major roles in the optimization and evaluates the first-order sensitives (first derivatives or directional derivatives) of the solution vector with respect to the perturbation parameters. In [21] and [23], both use a multi-point guidance algorithm that uses sensitivity analysis as a means of estimating the trajectories from an initial position of the lander to each candidate landing aim point.

3) *Learning Methods*

Learning Methods utilize neural networks to obtain the desired guidance and control laws. Contrary to the previous methods, the Learning Methods, need to learn what to do before an optimal guidance and control law can be achieved. Furthermore, rather than using mathematical expressions, of the motion of the lander, to determine the optimal law, they uses neural networks. This means that Learning Methods can be trained offline, and the obtained model can then be used for online calculations. Two learning methods that have been used for powered descent and landing are Deep Neural Networks and Reinforcement Learning.

Deep Neural Network:
Deep Neural Networks are a type of Artificial Neural Networks that have more than 1 hidden layer in them. Sánchez-Sánchez and Izzo, in [28], have trained a Deep Neural Network in a supervised manner on the optimal state-action pairs obtained via an indirect method. Hoowever, in their work, they assumed that perfect information on the spacecraft state. Altough, this train the network using supervised learning, in most real-life applications it would be impossible to get perfect information on the spacecraft state.

Reinforcement Learning:
Reinforcement Learning, instead of using supervised learning to learn, it learns by interacting with the environment. Thus it doesn't need to have per-obtained optimal state-action pairs, as for Deep Neural Networks, but instead it aims to learn them. In a reinforcement learning method, an agent (in this case the lander) interacts (i.e. performs actions) with the environment it is in, as a result of its interactions, it will receive from the environment the next state that it is going to be in, and a reward for having moved into that next state. The aim of a Reinforcement Learning method is to maximise the rewards that it receives from the environment.

Furfare and Linares [9], have used a reinforcement learning technique called Value Iteration to integrate ZEM/ZEV with a waypoint selection policy as a function of the current state of the lander during the powered descent phase. Jiang et al. used a reinforcement learning technique to determine a hand-over state in an Entry and Powered Descent guidance problem. The hand-over state is the state in which the lander has to switch from entry phase to powered descent phase.

Recently, due to the fact that neural networks are capable of solving NLP problems just through learning, reinforcement learning techniques have begun to be incorporated with neural networks to create a set of technique called deep reinforcement learning. Gaudet et al. [10] have implemented deep reinforcement learning methods as solutions to the guidance and control problem. In their work, they use a Reinforcement Learning technique called PPO to develop an integrated guidance and control system. Their technique works by taking the lander's sensors information as input and giving out a thrust command to the rockets of the lander, such that the lander can land with minimal error at a pre-determined location.

III. Methodology

A. *Model*

The model presented in this paper is a Recurrent Reinforcement Learning Network, a Reinforcement Learning model that utilises recurrence to make use of information extracted about previously observed states to mitigate the problem of partial observability. The base Reinforcement Learning algorithm used for this project is the Proximal Policy Optimization algorithm. The whole network is composed of 2 hidden fully connected layers, the first one of which is the recurrent layer of the network; an input layer, which consists of the observations received from the environment; and lastly the output layer, which is the same size as the action space of the environment. The representation of the model is shown in Fig 1. The way the network works, is that at every time-step t_n, the network receives as inputs the observations of the state that the agent is in, and the output of the Hidden Recurrent Layer at the previous time-step t_{n-1}; these two inputs are then combined into one singular vector. Once the inputs are combined together, the resulting vector is then forwarded through the network and it produces an output. In the case that the agent has a discreet action space, the output layer will have as many nodes as there are possible actions. If instead, the agent has a continuous action space, the number of nodes will depend on the formulation of the agent (In the model for this project the output layer consists of 2 nodes). For the recurrent node, a simple recurrent architecture was used.

1) PPO

Proximal Policy Optimization [30] is a Reinforcement Learning algorithm that came out in 2017 and quickly became one of the most used algorithms. This is due to the fact that it is an easy-to-use, easy-to-implement algorithm and it has demonstrated state-of-the-art performance for many RL problems. The PPO algorithm belongs to the family of actor-critic methods. They are called actor-critic because they employ the use of 2 networks, an actor network and a critic network. The actor network is the one that has to learn the policy, while the critic network has to learn the value function. Actor-critic algorithms are on-policy algorithms, meaning that they try to modify the policy that it's being used for decision making.

In order to learn the policy, PPO uses the policy gradient method. Policy gradient methods work by determining an estimator of the gradient of the policy and using it with a gradient ascent algorithm. The general form of a gradient estimator is:

$$\hat{g} = \hat{\mathbb{E}}_t[\bigtriangledown_\theta log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{10}$$

where π is the policy being evaluated and $\hat{A}$ is the advantage function, which, gives an estimate of how good an action taken in a certain state is. The estimator can be obtained by differentiating the loss function Eq. 11.

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{11}$$

The problem with policy gradient methods, is that, if we take multiple optimization steps on L^{PG} using the same trajectory, it can lead to having destructively large policy updates. In order to remedy this problem, [28] proposes two methods. The first method, Eq 12, is called Clipped Surrogate Objective. It consists in clipping the probability ratio between $1-\epsilon$ and $1+\epsilon$. The motivation behind this method is to avoid having the probability ratio move away from 1 too much, doing this ensures that the new policy will not deviate too much from the current one.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{12}$$

Where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$$

The second method, Eq. 13, is called Adaptive KL Penalty. This method functions by using a penalty on KL divergence, and to adapt the penalty coefficient so that it is possible to achieve some target value of the KL divergence d_{targ} each policy update.

$$\begin{cases} L^{KPLEN}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t - \beta KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \\ d = \hat{\mathbb{E}}_t[KL[\pi_{\theta old}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \\ if\ d < d_{targ}/1.5,\ \beta \leftarrow \beta/2 \\ if\ d > d_{targ} \times 1.5,\ \beta \leftarrow \beta \times 2 \end{cases} \tag{13}$$

For this project, the Clipped Surrogate Objective method was used, because, the authors of [30] have tested both methods and concluded that the Clipped method has better performance than the Adaptive KL Penalty.

2) Training Methodology

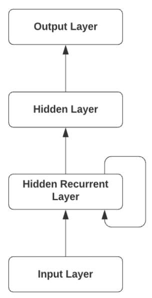
This section will include the techniques that were used in the training of the algorithm, so as to facilitate the reproduction of the results obtained in this paper. The training of the network was done by mostly following the training procedure explained in [30]. However, during the training of the model, instead of using the Stochastic Gradient Descent (SGD) (as suggested in [30]), the ADAM optimizer was used to adjust the learning rate for both the actor and critic networks and to perform gradient descent on both networks. Furthermore, in [30] the advantage function used was the Generalized Advantage Estimation (GAE), Eq. 14.

$$\begin{gathered} \hat{A}_t^{GAE(\gamma,\lambda)}(s_t, a_t) = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V \\ where\ \delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t) \end{gathered} \tag{14}$$

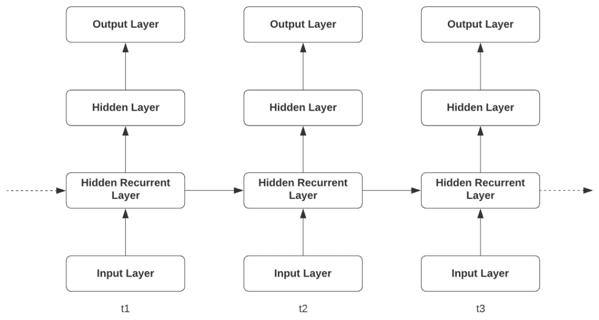
In this paper however, the advantage function shown in Eq. 15, is used. However, it is modified to use the values produced by the critic network ($\bar{V}^\pi(s_t)$) instead of $V^\pi(s_t)$, for training the PPO algorithm.

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{15}$$

The basic PPO algorithm that was used for training the model is shown in Alg. 2. For the sake of the actual implementation, the negative of the PPO-Clip objective is used. This is due to the fact that the optimizer used, only



(a) Folded RLL



(b) Unfolded RLL

Fig. 1. Recurrent Reinforcement Learning Model

allows for the minimization of an objective function. Thus, by taking the negative of the PPO-Clip objective, we can ensure that the steps taken are in the correct direction.

Algorithm 2 PPO training loop

```
1: **Input:** initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: **for** $k = 0, 1, 2, ...$ **do**
3:     Collect set of trajectories $D_k = \tau_i$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
4:     Compute returns $\hat{R}_t$, and advantage estimates $\hat{A}_t$ based on the current value function $V_{\phi_k}$
5:     Update the actor network by maximising the PPO-Clip objective:
       $$\theta_{k+1} = \arg \underset{\theta}{max} \underset{s,a \sim \pi_{\theta_k}}{E} [min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}, g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)))]$$
6:     Update the critic network by minimizing the mean-square error:
       $$\phi_{k+1} = \arg \underset{\phi}{min} \frac{1}{|D_k|T} \sum_{\tau \in D_k} \sum_{t=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2$$
7: **end for**
```

The model was trained using a learning rate of 0.01 for both 250,000 total timesteps and 500,000 total timsteps. Every batch of data used for training would contain the data of maximum 5619 total timesteps, collected from episodes of maximum length 500 timesteps. After a batch of data is collected, it is used 5 times consecutevly to update the networks. Before every iteration of network updates, the return $Q^\pi(s_t, a_t)$ is calculated using Eq. 16, for every episode collected in the batch. In equation 16, r_t is the reward received at timestep t. Wile, the λ^t is the discount factor that determines the weight of rewards, and it has to be between 0 and 1. At a value of 0 the algorithm would consider only the reward at that state, when the factor aproaches 1 it will consider future rewards as well, and it will incentivise the algorithm to strive for long term high rewards. For the implementation of this model a discount factor of 0.95 is used. In order to find the best hyperparameters, a trial and error approach was used. By including a validation function in the training loop, it was possible to analyze the performance of the algorithm and adjust the hyperparameters as needed. In reinforcement learning, validation is done by using the learned model in variations of the environment that differ from those used for training. This is because the algorithm has to be able to perform well even in environments that are different from those used in training. For this project, the training set of environments is given by seeds [20, 55, 137, 254, 329, 425, 567, 651, 739, 892, 946], while the validation set of environments is given by the seeds [33, 77, 111, 222, 333, 444, 555, 666, 777, 888, 999]. For each episode that is run, the triaining and validation seeds are randomly picked from their respective lists.

$$Q_\pi(s, a) = E_\pi[\sum_{t=0}^{T-1} \lambda^t r_t | s_t = s, a_t = a] \tag{16}$$

3) Lander Simulation

The framework used for the training of the RRL is the OpenAI Gym library, a library that gives access to various environments for developing reinforcement learning algorithms. The environment used for this paper is the LunarLanderContinuous-v2, shown in Fig 2, which is based on the Box2D simulator. The goal in this environment, is to land a 2D lunar lander on a landing pad. In every episode, the landing pad is always at coordinates (0,0), while the lander always spawns above the landing pad, the rest of the terrain is however, generated according to a seed.

From the environment, the agent, can receive a total of 8 observations: The x and y coordinates of the lander, the horizontal (v_x) and vertical (v_y) velocities, the orientation of the lander (θ), the angular velocity of the lander (v_θ) and two

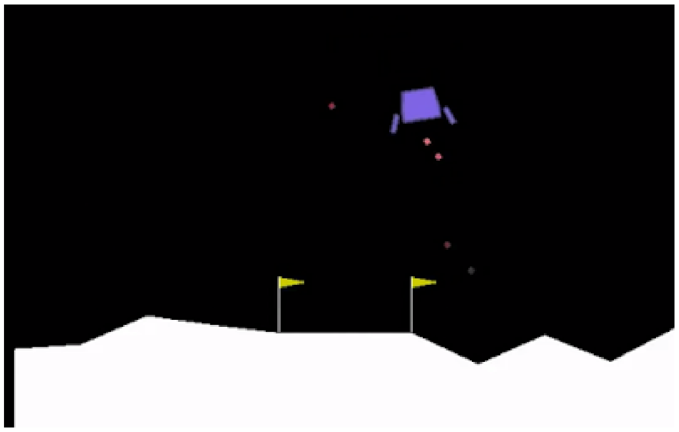


Fig. 2. OpenAI Gym Lunar Lander Environment

boolean values, one for the left leg and one for the right leg, representing whether the legs are touching the ground or not.

The landers coordinate are taken with respect to the landing pad. The x coordinate is positive when the lander is on the right of the landing pad and negative when it is one the left. The y coordinate it positive when above the landing pad, and negative when below it.

In this environment, the lander, has 2 continuous actions available, both of them accept values from -1 to +1. The first of the two actions controls the main engine, -1 to 0 the engine is off, 0 to +1 is the amount of throtle, 50% to 100%. Engines can't work below 50% power. The second action, controls the lateral engines, -1 to -0.5 fire left engine, +0.5 to +1 fire right engine and -0.5 to +0.5 off. The fuel available to the lander is infinite, however, the firing of the main engine subtracts 0.3 points each frame from the reward.

The reward that the agent receives at each timestep is dictated by the landers attitude and position, it is defined to be:

$$\begin{aligned} Rewards(s_t) = -100 \times \sqrt{x^2+y^2} - 100 \times \sqrt{v_x^2+v_y^2} \\ - 100 \times |\theta| - 0.30 \times P_m - 0.03 \times P_s \\ + 10\ (if\ right\ leg\ touches\ the\ ground) \\ + 10\ (if\ left\ leg\ touches\ the\ ground) \end{aligned} \tag{17}$$

where P_m is the main engine's power, P_s is the side engines' power. Furthermore, at the end of each episode, if the lander has crashed or flown out of the frame, then the agent receives -100 points. While if the lander comes to rest, the agent receives +100 points.

IV. Results

A. Training

The episodic rewards obtained during the training and validation stage of the development have been plotted into graphs and are shown in Figure IV-A. From the two plots it is already possible to see that the algorithm starts off with a very inconsistent policy, and as the training goes on the policy becomes better and much more stable. The reason for having trained two models one for 250,000 timesteps and the other for 500,000, is because during training, the agent showed a particular learning pattern that happened quite often. The pattern consisted in the model learning how to float rather than land after about 300,000 timesteps had been completed. Even in the models that learned to land after 300,000 timesteps, the lander would still come down slowly, such that in certain episodes it would reach the end of the episode before being able to land. However, it is important to note that altough the lander would come down slowly, it did so very stably, and with very minimal lateral displacement.

B. Testing

For the testing of the model, the actor networks have been made to run for a total of 1000 episode, the rewards collected by each network were then analysed, so as to extract the episodes with the mean, median, maximum and minimum reward. Furthermore, the landing accuracy of the models is extracted by calculating the percentage of episodes that end with the lander at rest on the landing pad. Lastly, the average number of timesteps taken to complete the episode (with the maximum being 1000) is reported as well. The results obtained are presented in Table I, and II for the 250,000 and 500,000 timesteps models respectively.

TABLE I
Rewards and Accuracy of the 250,000 timesteps model

Mean Reward	34.87
Median Reward	56.27
Min Reward	-322.51
Max Reward	273.96
Landing Accuracy	49%
Mean Timesteps	660

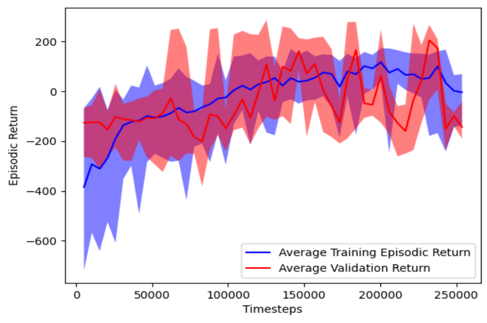
TABLE II
Rewards and Accuracy of the 500,000 timesteps model

Mean Reward	-49.75
Median Reward	-48.79
Min Reward	-252.02
Max Reward	63.49
Landing Accuracy	0%
Mean Timesteps	993

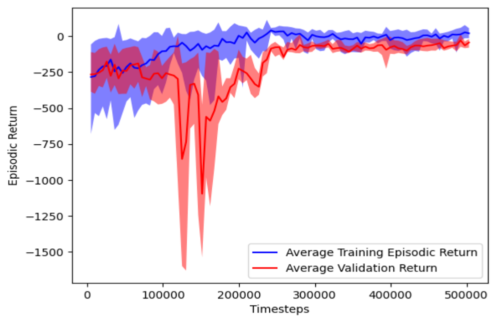
Although the statistics shown in the tables are a good indication of the performance of the two models, one thing that they don't show is the attitude that the lander has during descent. The 250,000 timesteps model, although it shows better performance, during the descent phase, it has a lot of lateral displacement and it is much less stable compared to the 500,000 timesteps model. The 500,000 timesteps model, on the other hand, although the results show a landing accuracy of 0%, there were plenty of times that the episodes finished with the lander on the landing pad, however, because the lander wouldn't come to rest on it they didn't count as successes. Furthermore, the 500,000 timesteps model also showed a much more stable attitude, with a much lower descent speed and less lateral displacement.

V. Discussion

The fact that the 250,000 timesteps model showed better performance, compared to the 500,000 timesteps one, could be due to the fact that during the training of the 500,000



(a) Plot of the episodic rewards obtained during the 250,000 timestep training



(b) Plot of the episodic rewards obtained during the 500,000 timestep training

Fig. 3. Testing and Validation Rewards Plots

timesteps model, it might have encountered series of rewards that discouraged it from having a faster landing speed and coming to rest, instead opting for a slow descent speed and no resting policy. This is further backed up by the fact that in the training and validation plots, the plot for the 500,000 timesteps model shows a couple of dips in the validation rewards. These dips happen just before the reward's variance decreases significantly, which further supports the fact that they influenced the direction of the training.

While the 250,000 model showed better performance, the 500,000 model showed a much better attitude control. The better attitude control could be a consequence of the recurrent node added to the network. This is because the recurrence, which helps the network understand the underlying systems dynamics better, should help with the stability of the agent by giving the agent a better idea of the state of the environment.

Although neither of the models showed a performance good enough for them to be considered as usable methods, overall, the performance of both models was satisfactory, whether its the statistical performance of the 250,000 timesteps model or the attitude control of the 500,000 model. Because of this, the fact that the technique is relatively easy to implement and that the forward network calculations can be done quickly enough, the RRL technique presented in this paper shows that it has potential for being a good controller in real life scenarios.

VI. Conclusion

In this paper, an Integrated Guidance and Control system based on a Recurrent Reinforcement Learning technique has been presented, and an analysis of related techniques has been performed. The analysis of related techniques, shows that in theory, using recurrence in a reinforcement learning technique, should help it perform better in real life scenarios in which the environment is only partially observable. The results obtained from the experiments performed further support the theory and show that the recurrence helped the technique to produce stable trajectories for the powered descent and landing phase.

Overall, although there is still a lot of work that can be done to improve the technique, the results show that it can achieve satisfactory performance and that there is potential for this type of technique to be considered as a good alternative to the currently used system for guidance and control.

VII. Future Work

There is still much work that can be done to improve the developed RRL model. One direction that the research could be taken in, is to try and adapt the objective function and training methodology to be better suited for the recurrent aspect of the algorithm.

The power of this algorithm is in its ability to help alleviate the partial visibility problem, that comes with almost all real life RL problems, by having a recurrent layer. Therefore, by using a simplistic environment like the LunarLanderContinuous-V2, it is harder to evaluate the effect that the recurrence has on the PPO algorithm. Thus, to better leverage the strengths of the model, it should be trained in a more complex environment.

As stated in sectiion II.A, some researchers [29], [17] have explored the use of RNN as a space state model to derive hidden states of the dynamics of the environment, as a means of recovering a Markovian state space. This could be an interesting direction in which to further develop the RRL model. Furthermore other changes to the RNN part that could be explored, is the use of other RNN techniques, such as LSTMs.

References

[1] Behçet Açıkmeşe and Lars Blackmore. Lossless convexification of a class of optimal control problems with non-convex control constraints. *Automatica*, 47(2):341–347, 2011.

[2] Behcet Acikmese and Scott R Ploen. Convex programming approach to powered descent guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 30(5):1353–1366, 2007.

[3] Saud Almahdi and Steve Y Yang. An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications*, 87:267–279, 2017.

[4] Bram Bakker. Reinforcement learning by backpropagation through an lstm model/critic. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 127–134. IEEE, 2007.

[5] Pieter Bram Bakker. The state of mind reinforcement learning with recurrent neural networks.

[6] Lars Blackmore, Behçet Açıkmeşe, and John M Carson III. Lossless convexification of control constraints for a class of nonlinear optimal control problems. *Systems & Control Letters*, 61(8):863–870, 2012.

[7] Xin Du, Jinjian Zhai, and Koupin Lv. Algorithm trading using q-learning and recurrent reinforcement learning. *positions*, 1:1, 2016.

[8] Behrouz Ebrahimi, Mohsen Bahrami, and Jafar Roshanian. Optimal sliding-mode guidance with terminal velocity constraint for fixed-interval propulsive maneuvers. *Acta Astronautica*, 62(10-11):556–562, 2008.

[9] Roberto Furfaro and Richard Linares. Waypoint-based generalized zem/zev feedback guidance for planetary landing via a reinforcement learning approach. In *3rd International Academy of Astronautics Conference on Dynamics and Control of Space Systems, DyCoSS*, pages 401–416, 2017.

[10] Brian Gaudet, Richard Linares, and Roberto Furfaro. Deep reinforcement learning for six degree-of-freedom planetary powered descent and landing. *arXiv preprint arXiv:1810.08719*, 2018.

[11] Carl Gold. Fx trading via recurrent reinforcement learning. In *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.*, pages 363–370. IEEE, 2003.

[12] Denise Gorse. Application of stochastic recurrent reinforcement learning to index trading. ESANN, 2011.

[13] Yanning Guo, Matt Hawkins, and Bong Wie. Waypoint-optimized zero-effort-miss/zero-effort-velocity feedback guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 36(3):799–809, 2013.

[14] Matthew W Harris and Behçet Açıkmeşe. Lossless convexification of non-convex optimal control problems for state constrained linear systems. *Automatica*, 50(9):2304–2311, 2014.

[15] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[16] Allan R Klumpp. Apollo lunar descent guidance. *Automatica*, 10(2):133–146, 1974.

[17] Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: a hybrid approach. *arXiv preprint arXiv:1509.03044*, 2015.

[18] Xinfu Liu. Fuel-optimal rocket landing with aerodynamic controls. *Journal of Guidance, Control, and Dynamics*, 42(1):65–77, 2019.

[19] Ping Lu and Xinfu Liu. Autonomous trajectory planning for rendezvous and proximity operations by conic optimization. *Journal of Guidance, Control, and Dynamics*, 36(2):375–389, 2013.

[20] Lin Ma, Zhengyu Song, and Zhijiang Shao. Simultaneous trajectory optimization framework for lunar ascent with terrain. In *2016 IEEE Aerospace Conference*, pages 1–10. IEEE, 2016.

[21] Lin Ma, Kexin Wang, Zhijiang Shao, Zhengyu Song, and Lorenz T Biegler. Trajectory optimization for planetary multi-point powered landing. *IFAC-PapersOnLine*, 50(1):8291–8296, 2017.

[22] Lin Ma, Kexin Wang, Zuhua Xu, Zhijiang Shao, Zhengyu Song, and Lorenz T Biegler. Trajectory optimization for lunar rover performing vertical takeoff vertical landing maneuvers in the presence of terrain. *Acta Astronautica*, 146:289–299, 2018.

[23] Lin Ma, Kexin Wang, Zuhua Xu, Zhijiang Shao, Zhengyu Song, and Lorenz T Biegler. Multi-point powered descent guidance based on optimal sensitivity. *Aerospace Science and Technology*, 86:465–477, 2019.

[24] Danylo Malyuta, Michael Szmuk, and Behcet Acikmese. Lossless convexification of non-convex optimal control problems with disjoint semi-continuous inputs. *arXiv preprint arXiv:1902.02726*, 2019.

[25] Yuanqi Mao, Michael Szmuk, and Behçet Açıkmeşe. Successive convexification of non-convex optimal control problems and its convergence properties. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 3636–3641. IEEE, 2016.

[26] J Meditch. On the problem of optimal thrust programming for a lunar soft landing. *IEEE Transactions on Automatic Control*, 9(4):477–484, 1964.

[27] John Moody and Lizhong Wu. Optimization of trading systems and portfolios. In *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, pages 300–307. IEEE, 1997.

[28] Carlos Sánchez-Sánchez and Dario Izzo. Real-time optimal control via deep neural networks: study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5):1122–1135, 2018.

[29] Anton Maximilian Schäfer. Reinforcement learning with recurrent neural networks. 2008.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.

[31] Michael Szmuk and Behcet Acikmese. Successive convexification for 6-dof mars rocket powered landing with free-final-time. In *2018 AIAA Guidance, Navigation, and Control Conference*, page 0617, 2018.

[32] Michael Szmuk, Behcet Acikmese, and Andrew W Berning. Successive convexification for fuel-optimal powered landing with aerodynamic drag and non-convex constraints. In *AIAA Guidance, Navigation, and Control Conference*, page 0378, 2016.

[33] Michael Szmuk, Utku Eren, and Behcet Acikmese. Successive convexification for mars 6-dof powered descent landing guidance. In *AIAA Guidance, Navigation, and Control Conference*, page 1500, 2017.

[34] Daniel R Wibben and Roberto Furfaro. Integrated guidance and attitude control for pinpoint lunar guidance using higher order sliding modes. In *22nd AAS/AIAA Space Flight Mechanics Meeting*, pages 1369–1382, 2012.

[35] Yabin Zhang and Tingting Yang. An integrated trajectory guidance and attitude control law with enhanced anti-disturbance capability for mars pinpoint landing. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 840–846. IEEE, 2017.

[36] Yao Zhang, Yanning Guo, Guangfu Ma, and Tianyi Zeng. Collision avoidance zem/zev optimal feedback guidance for powered descent phase of landing on mars. *Advances in Space Research*, 59(6):1514–1525, 2017.

[37] Liuyu Zhou and Yuanqing Xia. Improved zem/zev feedback guidance for mars powered descent phase. *Advances in Space Research*, 54(11):2446–2455, 2014.

MSc Project - Reflective Essay

Project Title	Recurrent Reinforcement Learning for Lander Control during Powered Descent and Landing
Student Name	Marco Anselmi
Student Number	160623970
Supervisor Name	Dr. Angadh Nanjangud
Programme of Study	MSc Artificial Intelligence

For my MSc Project, I decided to develop an Integrated Guidance and Control System (IGCS) using a Reinforcement Learning (RL) approach called Recurrent Reinforcement Learning (RRL). The Recurrent Reinforcement Learning approach sees the combination of Reinforcement Learning and Recurrent Neural Networks (RNN) as a means to solving a Reinforcement Learning problem of partial visibility of the environment. I was especially interested in researching the topic of Reinforcement Learning following my undertaking of Dr Paulo Rauber's "AI in Games" module, half of which was on Reinforcement Learning, in my first semester at Queen Mary University of London. Having undertook that module, I also had the confidence in being able to create the model I had set out to do. I completed the project having created a Proximal Policy Optimization neural network, with the first one of its hidden layers being a recurrent layer. The model works by taking as input the observations received from the Lunar Lander environment, of the OpenAI Gym library, and outputting a commanded thrust for each of the engines of the lander model.

ANALYSIS OF STRENGTHS AND WEAKNESSES

The principal strength of Recurrent Reinforcement Learning Techniques is that they can learn to approximate a Markovian state space from a Partially Observable environment. When an environment is fully observable, it can be said that the current observation *is* the current state of the environment. This means that an environment's response is depended only on the current observation and the current action. Because of this particular property, RL algorithms can achieve optimal control by just simply mapping observations to actions. However, if the environment is only partially observable, then the environment's response is not based on only the observations anymore, but there are some "hidden" factors that influence it as well. Because of this, creating a mapping between observations to actions becomes problematic, and the performance of the agent controlled by the algorithm becomes unstable. The use of neural networks in RL has already somewhat mitigated this problem. This is because neural networks are capable of storing some information regarding the system dynamics in their weights and biases. However, even with the inclusion of neural networks, real life control problems are still a challenging problem. Thus, in comparison to other RL techniques, the RRL technique should theoretically be able to perform better in real life situations.

Other strengths of the Recurrent Learning Technique come from the RL technique used to make the RRL model. For this project, the Proximal Policy Optimization (PPO) was used as the base model, therefore, the RRL model for this project has the advantage of being relatively easy to implement compared to other actor/critic models such as Trust Region Policy Optimization (TRPO), while having better performance. This comes from the use of the objective function. The PPO objective function, Eq 1, allows for better performance by clipping the probability ratio such that every step taken, in the optimization of the model, doesn't make the policy deviate too much from the previous one [7].

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)] \quad (1)$$

One of the major weaknesses of the model, is a weakness that affects all RL models. This weakness is the reliance on the environment. Because RL algorithms need to interact with an environment to learn the mapping between observations and actions, it means that what the algorithm can learn is dependent on the environment. Thus, even if the model learns a good policy during training, the learned policy might not be usable in real life/actual scenario if the environment that it was trained in doesn't reflect the actual environment it should be used in. Another problem, which comes from general RL algorithms, is the implementation of hard state constraints. In order to set hard state constraints, it takes manual adjustments in a trial-and-error manner. Although some research [1] has been done in solving the problem, most approaches still use a trial-and-error manner. Even though for this project the rewards function didn't need to be set, as I was using the Lunar Lander environment of the OpenAI Gym, which comes with a function for giving rewards, the problem is still present for the RRL technique.

Another weakness that came up during the development of the RRL model, is that there is very little documentation of such techniques. Although, this is not a weakness of the model itself, but rather of the difficulty of implementation. Most of the research that has been done for RRL techniques, was done in the field of finance, and even if some of the work done in the field of finance can be transferred over to the field of guidance and control, there are sill a lot of aspects of the finance models that are useless for the guidance and control models. This is due to the fact that the underlying problems, that they are trying to solve, are very different. At the initial stages of the project, only two papers that tackled the use of RRL in the field of control theory were found. While, later on in the project development some more documentation was found, it was still very minimal, to the point that at the end of the implementation I only managed to find five papers [6], [5], [4], [2] and [3]. This, forced me to change the methodology with which I was doing research to find possible improvements to be implemented and solutions to problems that I was facing during the development. Ultimately, it also led me to slightly change the original overall design that I had for the model, as due to lack of documentation and lack of time, I was not able to make the original model work.

PRESENTATION OF POSSIBILITIES FOR FUTURE WORK

The main follow on work would be to try and integrate the two techniques, Reinforcement Learning and Recurrent Neural Networks, in a much better way. In the proposed model, the integration of the two techniques was done by simply including a recurrent node instead of the first hidden layer of the PPO network.

The first possibility for a better integration, is to continue the development of my original plan. In the original plan, a Recurrent Neural Network was to be used as a State Space Model (SSM), which can be used to learn the dynamics of a system. Then, once the SSM learnt the underlying dynamics of the environment, the output of the hidden layer of the network would be used as the input to the PPO actor network. The reasoning behind this technique is that by using the hidden layer of a SSM, it is possible to reconstruct a Markovian state space of the environment from the available observations. This is because, in a partially observable environment, there are some hidden factors, that are not observable, which influence the responses of the environment to the agent's actions. Thus, if the Recurrent Neural Network is an SSM, it is possible to use the network's hidden layers in place of the environment hidden factors. As a matter of fact, most techniques that use RRL in the field of guidance and control, use a methodology like this, rather than making the actor network recurrent.

Another possibility for a better integration, would be to look at the objective function used. Even though there is no research that has been done in this area, it would be interesting to explore the effect that the added recurrence has on the ability of the model to learn, while using the original PPO objective function versus a modified version. Although, using the original PPO objective function should be fine, it is possible that by adding a recurrent node the objective function struggles to allow the model to form a good mapping between the inputs and actions. This could be due to the fact the input is not only composed of observations, but now it also contains a memory of the previous timestep, making the mapping not a mapping from observations/state to actions, but a mapping between an input to an action.

As stated before, Reinforcement Learning algorithms are highly dependent on the environment for what they can learn. In this paper the environment used is the LunarLanderContinuous-V2, which is provided by the OpenAI Gym library. Although, this is a good initial environment, it is rather simple, with a dynamical system that is not overly complex. This is because, it is emulating a lander in the descent and landing phase on the moon. The moon itself has a very simple environment, since most of the aerodynamical influences on the lander can be ignored, and with the lander being in the descent and landing phase, variations in the gravitational acceleration can be ignored, as the lander would be close enough to the surface where they would be almost non-existent. Furthermore, it being a 2-D simulation, makes the environment less complex, as it constrains it to a 3 Degrees of Freedom problem, and it also makes it so that the policy learned cannot be used in a real-life scenario. Because of these factors, implementing the algorithm in a different, more complex environment, it would allow for a more appropriate analysis of the technique, and it would enable us to fully exploit the use of the recurrence of the model.

Last but not least, a possible future direction of research, it's in the exploration of the RRL technique with different RL algorithms and RNN methodologies. This paper only analysed the use of a basic RNN node in a PPO algorithm. However, there are multiple RL algorithms, each with its own advantages and disadvantages. Furthermore, RNNs also have different types of nodes that can be use, such as the Long-Short Term Memory node, which helps with retaining information from timesteps further back in the past. By exploring RRL techniques with various RL algorithms and RNN methodologies, it might be possible to find common strong points and common weak points. Finding those points in common would allow researchers to have a much better understanding of the strengths and weaknesses of RRL, thus having a more general idea of which direction research should continue in.

CRITICAL ANALYSIS OF THE RELATIONSHIP BETWEEN THEORY AND PRACTICAL WORK

During the development of this project, I have had the opportunity of realising what the difference between theory and practical work is. During the initial period of research and literature review, I focused on finding a technique that I could work with. I focused my research on reinforcement learning algorithms and trying to find the one that would suit the problem I was tackling the best. During my research I learned that there are two types of RL algorithms, value function based ones and policy gradient based ones. According to the knowledge that I had gained while researching, the only category of algorithms that I could use for my problem was the policy gradient ones. This is because, the problem that I was tackling had continuous action and state spaces, meaning that the values of the observations and actions are continuous. Since value function based ones need to be able to evaluate every possible action for them to work, having continuous action spaces makes it impossible for them to do so, thus rendering them useless in cases such as the one of my problem. Later on during the development of the project, I then discovered, that by tweaking the way the agent produces actions and the way the environment receives them, it becomes possible to use value function based algorithms for problems that are originally with continuous action and state spaces. This was the first I came across the relationship between theory and practical work during the development of my project.

The second time was during the experiments I ran for the models I had created. At first, I had a different type of model designed, which, in theory should have worked better than the one I have created now. This original model was composed of two different parts, a State Space Model and a PPO algorithm, which used the SSM's hidden state as the input. This methodology should in theory have worked fine, as the PPO would just learn a mapping between the hidden states and actions, rather than observations and actions. However, when it came to the actual implementation of it, I couldn't get the PPO algorithm to learn anything, neither the actor nor the critic network showed any improvement. Later in the development, once I came up with the final version of the model, there were still some discrepancies between theory and practical work. In theory, when training a RL algorithm, it needs to train itself on a lot of timesteps. In practice, however, my model trained fine until around it had gone through 300,000 timesteps, after which, the performance of the model would start to slowly decrease and it would learn to float rather than land. However, in both the original model and the final model, the theory regarding RNNs ability to approximate dynamical system was applied successfully. This was shown by the fact that the SSM was able to accurately predict what the next state was going to be, and the fact that the final model managed to achieve a policy that creates stable trajectories.

Overall, theory is based on the human understanding of how things work or should work. Thus, when the understanding of how things work or of the theory itself is flawed, the theory is not going to work in practice. Also, the more we practice the more our understanding of how things work increases, thus improving the theory.

AWARENESS OF LEGAL, SOCIAL, ETHICAL ISSUES AND SUSTAINABILITY

The main legal, social and ethical consideration for this project, is in the case it is used in a real-life situation. During the training, no particular legal, social or ethical considerations needed to be taken into account. This is because no other humans were involved in the making of the project, thus removing most ethical and social issues that could arise, other than the obvious ones such as fraud and plagiarism. In terms of legal issues, all software used in the project was available for use without the need of special permissions from the creator, thus, no copyright laws were broken and no other laws needed to be taken into consideration during the making of the project.

Possible issues might arise if the project were to be used in a real-life scenario. Because the algorithm is used to control a rocket, there is the ethical issue that in the case of a failed landing, it could cause the death of the people on board the lander. In terms of legal issues, it could lead to problems with liability for damage laws and environmental preservation law.

The training of the model takes a really long time, which can cause sustainability issues. The shortest of the trained models took 2 and a half hours, and the longest took 12 hours of training. Because training AI models uses a lot of resources of a computer, it also means that the power consumption of the computer increases during training. Therefore, training the algorithm for such long times can have a carbon footprint due to the high energy consumption, therefore, it should be subject to environmental sustainability ethics.

REFERENCES

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

[2] Bram Bakker. Reinforcement learning by backpropagation through an lstm model/critic. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 127–134. IEEE, 2007.

[3] Pieter Bram Bakker. The state of mind reinforcement learning with recurrent neural networks.

[4] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 aaai fall symposium series*, 2015.

[5] Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: a hybrid approach. *arXiv preprint arXiv:1509.03044*, 2015.

[6] Anton Maximilian Schäfer. Reinforcement learning with recurrent neural networks. 2008.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.